Benczúr András
2008.

Adatmodellezés

Ismétlés: a relációs adatmodell

Az adatmodellezés céljai:

(i) - a valóság, a „valós világ" leírása, összetevőinek, tényeinek reprezentálása, összefüggések, megszorítások megfogalmazása

(ii) - műveletek végrehajthatósága, a valós viszonyoknak a modellben való tükröződését megőrizve, információ szolgáltatása (számítógépre vihető módon)

(iii) - hatékony implementálhatóság, nagy mennyiségű adattal való gazdálkodás. (erőforrások)

Séma és előfordulás:

Séma: az adatbázisban előforduló adatok típusa; absztrakt adattípusok névvel való hivatkozással.

Előfordulás: az egyes típusoknak a sémában adott nevű konkrét előfordulásai, a konkrét adatértékek.

Az (i) szintnek az információs rendszerek tervezésénél használt logikai, fogalmi adatmodellek felelnek meg. Legismertebbek:

- Egyed-Tulajdonság-Kapcsolat modell (Entity-Relationship : ER modell)
- Az UML adatleíró része (Unified Modeling Language)
- Az ODL : objektum-adatmodell
- Legújabb: XML alapú, ontológiák, szemantikus Web

A (ii) szinten az adatokkal végzett műveletek szerepe jelenik meg, ez válik fontossá.

A legelterjedtebb modell jelenleg is a <u>relációs modell</u>.

Továbbfejlődése: az objektum-relációs modell, absztrakt adattípusokkal kiterjesztett reláció.

Legújabb modellek: a félig-strukturált adatmodell, dokumentum-modell: szabványossá válik az XML adatmodell is ABKR.

A (iii) szempont, a hatékony implementálás, nagy mennyiségű adattal való gazdálkodás jelenleg a <u>relációs modellre</u> alapul.

A reláció, mint absztrakt adattípus egyszerű, ezért hatékonyan implementálható.

Hogy néz ki a séma, előfordulás a relációs modellben:

<u>A Séma</u>:

Névvel ellátott eleme a <u>reláció</u>. A reláció <u>típusát</u> egy <u>sortípus</u> adja meg.

A <u>sortípus</u> két változat:

- név nélküli (attribútumnév nélküli) eset: <u>elemi értékekből álló n-es</u>
  $\langle$ érték$_1$, ..., érték$_n$ $\rangle$
- névvel ellátott eset: attribútumnevek
  $\langle$ Anév$_1$: érték$_1$, ..., Anév$_n$: érték$_n$ $\rangle$, <u>sor</u>

Tipikus megadása: <u>Relációséma</u>

Rnév (Anév$_1$, ..., Anév$_n$) (Konvenció: R(A,B,C,..))

Az attribútumokhoz értéktípus/értéktartomány tartozik: Dom$_1$, ..., Dom$_n$ (Domain)

<u>Reláció adatbázis sémája</u>: Véges sor relációséma: $R_1, R_2, \ldots, R_k$

SQL: CREATE TABLE utasítás.

Bui

Az Előfordulás:

Az $R(A_1, \dots, A_n)$ relációséma hoz a típusának megfelelő típusú véges sok sor az <u>előfordulása</u>.
Vizuálisan táblaként (táblázatként) ábrázolható.
Ez egyben egy jól ismert <u>dokumentumtípus</u>.
Két dimenziós tábla: Első sor: fejléc, attribútumok - oszlopnevek.

(Codd eredeti értelmezése: az előfordulás sorok <u>halmaza</u>, és az 1NF: atomi értékek, és minden sor különböző.

Lehet általánosabb értelmezés: sorok <u>multihalmaza</u>. Egy sor többször előfordulhat. Műveleteknél fontos különbséget eredményez! Az SQL SELECT eredménye az alapértelmezés szerint multihalmaz!

Sorok sorrendje nem számít.
Oszlopok <u>névvel együtt</u> felcserélhetők.

<u>Jelölés:</u> R sémájára $R(U)$, $U = \{A_1, \dots, A_n\}$ rokonos, ahol $A_1, \dots, A_n$ attribútumok.

R és előfordulása (az adatbázisban a jelenlegi, kurrens. előfordulás) közötti nem teszünk általában különbséget a <u>jelölésben</u>.

A reláció neve egyszerre utal <u>sémájára</u> és előfordulására.

(Ha szükséges, például függőségek tárgyalásánál az $I \in Instance(R)$ jelölést használjuk, ahol I az R séma egy előfordulása.)

Sor jelölése: - $t \in R$, t az R sora.

- $t[A_i]$, vagy $t(A_i)$, vagy $t.A_i$: mindegyik a t sor A_i attribútumhoz tartozó értéke.

- $t[X]$, ahol $X \subseteq U$: az $\{X\}$ attribútumhalmazon felvett értékei a t sornak.

Megszorítások, feltételek az előfordulásokra

(A séma szinten megadható megszorítások. Később még további részletek következnek!)

Kulcs: Az $R(A_1, \dots, A_n)$ relációsémához az $X = \{A_{i_1}, \dots, A_{i_k}\} \subseteq U$ attribútumhalmaz kulcs, ha R megengedett előfordulásaiban $t_1, t_2 \in R$ és $t_1[X] = t_2[X]$ esetén $t_1 = t_2$. (Vagyis $t_1[U] = t_2[U]$, $X \to U$ funkcionális függőség teljesül.)

Vagyis az X-en felvett érték egyértelműen meghatározza a teljes sort.

(SQL-ben: PRIMARY KEY
UNIQUE)

Hivatkozási épség

$R(A_1, \dots, A_n)$ és $S(B_1, \dots, B_m)$ között adható meg:

$X \subseteq \{A_1, \dots, A_n\}$ kulcs az R-ben, $X = \{A_{i_1}, \dots, A_{i_k}\}$

$Y = \{B_{i_1}, \dots, B_{i_k}\} \subseteq \{B_1, \dots, B_m\}$ idegen kulcs S-ben, FOREIGN KEY

azt azt jelenti, hogy

$s \in S$ esetén létezik $t \in R$, hogy

$s[Y] = t[X]$

SQL: CREATE TABLE utasításban:
PRIMARY KEY
FOREIGN KEY ... REFERENCES ...

Irodalom: Ullman-Widom: Adatbázisrendszerek Alapvetés 6.1. és 6.2 szakaszok.

Bemi

Műveletek relációkkal : a relációs algebra és az algebrai lekérdezések

A műveleteket előfordulásokon (táblákon) értelmezzük, majd kifejezéseket építünk, így kapunk lekérdezést.
A műveletek a válasz előállítási módját adják meg, azaz a relációs algebrai lekérdezés procedurális nyelv.
A műveleteket kis táblákon szemléletesen könnyen követhetjük.
(Csak az attribútum neveket használó modellre adjuk meg a műveleteket. A név nélküli eset kis mértékben eltér.)

Az alapműveletek:

$R \cup S$: azonos típusú relációk R és S, az unió a két sorhalmaz egyesítése

$R - S$: R és S azonos típusú; az R sorainak halmazából elhagyjuk az S-ben is előforduló sorokat. (Fontos tulajdonsága, hogy nem monoton!)
<u>Különbség</u>

$R \bowtie S$: <u>természetes összekapcsolás</u>;
legyenek R és S közös attribútumai $X = \{A_{i_1}, \dots, A_{i_k}\}$;
minden $t \in R$ és $s \in S$-hez, ha $t[X] = s[X]$, vesszük a két sor egymásra húzását. Az X-hez tartozó értékeket természetesen csak egyszer vesszük.

$\Pi_{lista}(R)$: <u>projekció</u>; $lista = \{A_{i_1}, \dots, A_{i_k}\} \subseteq U$, ahol $R(U)$ az R sémája.
Ha $t \in R$, vesszük a $t[A_{i_1}, \dots, A_{i_k}]$ sort.
Duplikátumokat kiszűrjük!

5

Bemi

$\sigma_{Feltétel}(R)$: <u>kiválasztás, szelekció;</u>

Feltétel: $A \Theta B$, vagy $A \Theta$ "a" atomi feltételekből felépített logikai ~~kifej.~~ formula, ahol

$A, B \in U$, és "a" konstans.

Θ : $=, \neq, \leq, \geq, <, >$ összehasonlítási.

Eredménye: ha $t \in R$ és t kielégíti a feltételt, akkor t kiválasztásra kerül az eredménybe.

<u>Átnevezés</u>: $\rho_{A \to B}(R)$: új relációt képez, átnevezve az A attribútumot B-re.

$\rho_{X \to Y}(R)$: általánosabb: az X attribútum-~~ma~~ halmaz elemeit helyettesíti Y elemeivel.

Ez a 6 alapművelet. (Nem redundáns rendszer, egyik sem fejezhető ki a többivel.)

Ezekből <u>építünk lekérdező nyelvet</u>: algebrai <u>kifejezéseket</u>

Kifejezés felépítése: ha E, E_1, E_2 kifejezések, akkor:

- A konstans reláció <u>kifejezés</u>. Értéke saját maga.
- R relációnév <u>kifejezés</u>, értéke: az adatbázisban szereplő <u>előfordulás</u>.
- Ha E, E_1, E_2 kifejezés, akkor $E_1 \cup E_2$, $E_1 - E_2$, $E_1 \bowtie E_2$, $\Pi_{lista}(E)$, $\sigma_{Feltétel}(E)$ és $\rho_{X \to Y}(E)$ kifejezés.

6

Bmi

Adott $\mathcal{R} = \{R_1, \dots, R_k\}$ adatbázis sémán az

E <u>relációs algebrai kifejezés kiértékelése:</u>

A kifejezés kiértékelő fáját alulról felfelé értékeljük ki. A levelek vagy konstans táblát, vagy adatbázis relációnevet tartalmaznak. A relációnevekhez behelyettesítjük az előfordulásokat, és a konkrét táblákkal végezzük el a műveleteket.

<u>Megjegyzések</u>: Az adatbázis-kezelő rendszerek az SQL lekérdezéseket általában algebrai kifejezésre fordítják, majd a kifejezést ekvivalens átalakításai hatékonyabban kiértékelhető alakra ~~tes~~ hozzák. Végrehajtási tervet készítenek, és azok közül választják a legjobbnak ígérkezőt. Egzakt optimalizálás nem lehetséges. Ennek oka, hogy ~~az algebr~~ a relációs algebrai kifejezések ekvivalenciája nem dönthető el.

Az algebrai kifejezések és az SQL SELECT:

Az alap SELECT utasítás az egyszerű

```
SELECT ..
  FROM ...
    WHERE
```

alak egy szelekció – összekapcsolás – ~~s~~ projekció kifejezésnek felel meg.

$R \cup S$ megfelelője: (SELECT ..) UNION (SELECT ..)

$S \cap R$: INTERSECT

$R - S$: EXCEPT (MINUS)

Irodalom: U-W könyv: 4.1. – 4.3. és 5.1. – 5.5. szakaszok

Beni

Fogalmi - Logikai adatmodellezés
(Magas szintű adatmodellek)

A „valós világ" leíró (adat)modelljei.

"Ideák" → "Magas szintű" tervezés → Relációs Adatbázis-séma
↓
Relációs
A B K R
Tipikus végeredmény

Műveletek a típusokon nem játszanak szerepet. Ezért logalmi modell.

Legelterjedtebb modell: az Egyed - Kapcsolat
angolul: Entity - Relationship (ER) modell.

Chen: 1975-ben javasolta.
(Évente van még mindig ER konferencia.)

Építő elemei: Fogalmak — névvel ellátott séma-elemek —
— Előfordulások

Entitás: más entitástól elkülönítheto, azonosítható.

Alap fogalmak (alaptípusok):

- Egyedhalmaz (Entityset) entitásokból, egyedekből áll. Az entitás az alapfogalom, az „idea".
- Tulajdonságok (Attribútumok, Attributes)
- Kapcsolatok (Relationships)

Egyedhalmaz: névvel ellátott halmaza (valamilyen szempontból) hasonló entitásoknak.
Előfordulása: az egyedhalmaz.
pl: Filmek, Stúdiók, Diákok, Tanárok, Kurzusok,..

Tulajdonságok: Az egyedeken tulajdonságok figyelhetők meg. A megfigyelés a tulajdonsághoz tartozó értéktartományból egy értéket rendel az egyedhez.

①

Bemi

Pl: hossz, név, élettar, ...

Az egyedhalmaz éppen az jellemzi, hogy milyen tulajdonságok figyelhetők meg a halmaz elemein. Ezek után már megadhatjuk az egyedhalmaz sémáját:

⟨Halmaznév⟩ (⟨Tulajdonságnév$_1$⟩, ... ⟨T név$_k$⟩)

/ Hasonlót a relációs-sémánál.)

<u>Kapcsolatok</u> Az egyedek között kapcsolatok figyelhetők meg. Egy <u>névvel</u> azonosított kapcsolat kapcsolatban álló egyedek halmaznevével jellemezhető. (Azt adja meg, milyen típusú entitások között áll fenn a névnek megfelelő kapcsolat.)

⟨Kapcsolatnév⟩ (Halmaznév$_1$, Halmaznév$_2$, ..., Halmaznév$_k$)

Előfordulása: k-asok az osztályok elemeiből.

<u>Grafikus ábrázolása:</u>

Egyedhalmaz : Halmaznév

Tulajdonság : Tulajdonság-név

Kapcsolat : Kapcsolat-név

Ezek az alapdiagram elemei.

<u>Séma előfordulása:</u> A halmazok benépesítése <u>egyedekkel</u>, minden tulajdonsághoz <u>érték</u> minden egyedre, kapcsolatok előfordulásai: konkrét, a halmazokban előforduló egyedek között létező kapcsolatok. (szorzatának részhalmaza)

(2)

Bemi

További modellezési lehetőségek, megszorítások megadása:

Bináris kapcsolatok:

Multiplicitási viszonyok:

- Sok az egyhez E-ből F-be

N:1 viszony, nyíl az 1 felé.

E minden eleme legfeljebb egy F-beli elemhez kapcsolódik.

- Egy az egyhez:

1:1 viszony

- Sok a sokhoz:

N:M viszony

Többszörös kapcsolatok:

Példa jelentése:
a G-be mutató nyíl azt jelenti, hogy egy E, F-beli párhoz legfeljebb egy G-beli elem kapcsolódhat.

Megjegyzés: Kapcsolatot is tekinthetünk entitásnak. Választás, döntés kérdése.

Például:

ekvivalens:

További Példák. U-W: Adatbázisrendszerek 2.2., 2.3.

A diagrammokban a kapcsolatokhoz szerep is rendelhető; a szerepet az élek címkéjével adjuk meg.

Kapcsolathoz is rendelhető tulajdonság:

Például

helyett

3

Bemi

Az isa hierarchia.

(Az U-W fordításában „az egy" - re fordították.)

Szerepe: speciális részhalmaz megadása. További tulajdonságai lehetnek, más kapcsolatban lehet résztvevő.

Jelentése:
Ha $e \in A$ akkor egyben $e \in B$ is teljesül.
Az A előfordulásai a B előfordulásainak részhalmaza.

Az isa hierarchia irányított aciklikus gráfot alkot! (Másik ábrázolás:

Kulcsok és hivatkozási épség

Az E egyedhalmazban a tulajdonságok egy K halmaza kulcs, ha E bármely két különböző e_1, e_2 egyede nem vesz fel a K minden elemén azonos értéket. Legalább egy K-beli tulajdonságuk eltér.

Kulcs-feltétel

Minden egyedhalmazhoz legyen kulcs.
Az isa hierarchiában ez azt jelenti, hogy a gyökérben legyen kulcs. (Az más lefelé a részhalmazokban is kulcs.)

Egyöntetűen kulcsokat külön jelöljük.
Jelölés: aláhúzással

(4)

Bemi

Hivatkozási megszorítások kapcsolatokban:

Kötelező részt venni egy kapcsolatban: kerek nyílvég

Kötelező kapcsolat, kell Stúdió egyed.

Jelentése: Tvez-hez kötelező a vezetett tanszék léte.

Lehet a kapcsolatok nyilaihoz alsó-felső korlátot adni: a nyíl végén szereplő egyedből legalább ennyi és legfeljebb annyi vehet részt az adott kapcsolatban.

Gyenge egyedhalmazok:

Ha egy egyedhalmaznak nincs közvetlen saját kulcsa: az gyenge egyedhalmaz. Ilyenkor más egyedhalmazok tulajdonságaival kiegészülve lesz kulcsuk.

Jelölés:

Gyenge egyedhalmaz:

, kettős vonal

Kapcsolata, ahonnan a kulcs egy részét kapja:

: kettős,

Sok-egy kapcsolat!

A saját tulajdonságok közül a kulcsban részt vevő: aláhúzva

Példák: U-W: 2.6.

(j)

Bem

Példa:

A kontrollt kulcsot ö rökül:
N : 1 - es viszonyban áll vele!

Példa:

A gyenge egyedhalmaz a kapcsolatban résztvevő többi egyedtől kapja a kulcsát.

Lekérdezés a relációs modellben

Alapelvek: kezdjük az erős egyedhalmazokkal. Mindegyik önmagában relációnak is tekinthető, kulcsa is van.

Halmaznév → Relációnév
Tulajdonságnév → Attribútumnév
Kulcs → Primary Key

(6)

Bemi

<u>isa</u> hierarchia: minden halmazhoz hozzávesszük a gyökér kulcsát. Erős halmazzá tesszük. Ezt képezzük relációra.

<u>Kapcsolat</u> reprezentálása: az <u>erős</u> osztályok egyedeit kulcsértékek reprezentálják.

A ~~kap~~ kapcsolatnak megfelelő relációt: az erős egyedosztályok kulcsaiból képezzük sorait. Az 1:N kapcsolatok funkcionális függőséget jelentenek.

<u>Gyenge</u> egyedhalmaz: hozzávesszük a külső egyedekből a szükséges tulajdonságokat – erős egyedhalmazzá tesszük. Az N:1 kapcsolatait az együtthal már reprezentálja.

Az egyedhalmaz egyedeit ezzel a kulccsal kiegészítve képezzük relációba.

Először tehát minden egyedhalmazt kulccsal látunk el, erőssé tesszük, így képezzük relációba.
Ha ezzel más reprezentálunk egy kapcsolatot, azt nem kell külön feltüntetni.

Egyedhalmaz kulcsa a kapcsolatban <u>idegen kulcs</u> lesz a relációs modellben.

Objektum-orientált modell: ODL
szerepel az U-W könyvben, együtt tárgyalja az EK modellel.

Újabban előtérbe került adatmodellezésben is az UML (Unified Modeling Language)

7

Bennő

Fogalmak megfeleltetése

UML	E/K modell
Osztály	Egyedhalmaz
Asszociáció	Bináris kapcsolat
Asszociáció-osztály	Tulajdonság kapcsolathoz
Alosztály	Isa hierarchia
Agreggáció	Sok-egy kapcsolat
Kompozíció	Sok-egy kapcsolat hivatkozási kényszerrel

Osztály diagramja:

Ezt nem használjuk az adatmodellezésnél!

Kulcs: PK jelöli (primary key)

Asszociáció:

Él két össze két összetevőt: Bináris, nevu van, mindkét irányban, multiplicitás jelöltel.

(8)

Bemi

m..n jelentése: a vonal másik végén álló O2 objektumhoz legalább m és legfeljebb n O1 típusú objektum tartozik.

A * , mint speciális érték korlátozás nélküli előfordulást jelent.

Speciális esetek:

1..1 : pontosan egy. Ez az alapértelmezés.

0..1 : legfeljebb egy

0..* : tetszőleges sok

1..* : legalább egy.

Lehet visszaható asszociáció:

Asszociáció – osztály : Az él közepére ráakasztunk egy osztályt:

Alosztályok

Felbontás alosztályokra lehet:

Diszjunkt – Átfedő

Teljes – részleges

9

Bemi

Agregátum és kompozíció

A sok - 1 kapcsolat speciális esetei:

Agregátum:

A végződés jelentése:
0 . . 1
multiplicitás

Kompozíció:

A végződés jelentése:
1 . . 1
multiplicitás

(10)

Bemi

Relációs adatbázisok tervezése, függőségek, normalizálás, megszorítások

Irodalom: Ullman - Widom: 3.5. — 3.9.

A jó adatbázis-séma kialakítása:
- anomáliák (beillesztés, törlés, módosítás) kiküszöbölése
- funkcionális függőségek elemzése
- normalizálás - dekomponálás

Funkcionális függőségek

Az $R(U)$ séma felett $X \to Y$ funkcionális függőséget jelöl, ahol $X, Y \subseteq U$. Ha $Y = U$, akkor kulcsfüggőség.

Az I előfordulás kielégíti $X \to Y$ -t, ha bármely $t_1, t_2 \in I$ sorokra, és ha $t_1[X] = t_2[X]$, akkor $t_1[Y] = t_2[Y]$.

Logikai jelölés: $I \models X \to Y$.

Függőségek implikációja, (logikai következmény)

Legyen $\mathcal{F}: \{ X_i \to Y_i \mid i = 1, \dots, k \}$ funkcionális függőségek véges halmaza.

Az $f: X \to Y$ funkcionális függőséget implikálja $\mathcal{F}$ (más szóhasználattal: f az $\mathcal{F}$ logikai következménye), ha minden I előfordulásra $I \models \mathcal{F} \Rightarrow I \models f$.

Vagyis, ha I kielégíti $\mathcal{F}$ minden függőségét, akkor I f-et is kielégíti.

Kérdés: el lehet-e dönteni az ~~van~~ $\mathcal{F} \overset{?}{\models} f$ implikációt? Hogyan dönthető el?

(1)

Bemi

1. Módszer : <u>X lezárása $\mathcal{F}$ szerint.</u>

Legyen $X \subseteq U$, $\mathcal{F}$: funkcionális függőségek halmaza. $\mathcal{F} = \{ X_1 \to Y_1, \dots, X_k \to Y_k \}$.

X lezártja $\mathcal{F}$ szerint:

$$X^{+\mathcal{F}} = \{ A \mid \mathcal{F} \models X \to A \}.$$

<u>Lemma</u>: $\mathcal{F} \models X \to Y \iff Y \subseteq X^{+\mathcal{F}}$.

(Az $\mathcal{F}$ akkor és csak akkor implikálja $X \to Y$-t, ha Y részhalmaza X $\mathcal{F}$-szerinti lezártjának.)

<u>Állítás</u>: Létezik hatékony algoritmus $X^{+\mathcal{F}}$ előállítására. (l: U-W. : 3.6.3. szakasz)

2. Módszer: Levezetési szabályok, axiomatizálás

<u>Armstrong - axiómák</u>

F1. Reflexivitási axióma:
$Y \subseteq X$ esetén $X \to Y$ mindig teljesül.

F2. Bővítési axióma:
Ha $X \to Y$ teljesül, és $V \subseteq U$ tetszőleges, akkor $XV \to YV$ is teljesül.

F3. Tranzitivitási axióma:
Ha $X \to Y$, és $Y \to Z$ teljesül, akkor $X \to Z$ is teljesül.

Levezetés: Az $X \to Y$ levezethető az $\mathcal{F} = \{X_i \to Y_i, i=1,\dots,k\}$ függőségi halmazból, ha az $\mathcal{F}$-beli függőségekből az F1, F2, F3 szabályok véges sokszori alkalmazásával megkapjuk $X \to Y$-t. Jelölés: $\mathcal{F} \vdash X \to Y$

<u>Tétel</u> Az Armstrong axiómák helyesek és teljesek.

Helyesek: $\mathcal{F} \vdash X \to Y \Rightarrow \mathcal{F} \models X \to Y$. (Bizonyítás nélkül!)

Teljesek: $\mathcal{F} \models X \to Y \Rightarrow \mathcal{F} \vdash X \to Y$.

②

Bemi

Többértékű függőség:

$\mathbf{R}(U)$ séma felett, $X, Y \subseteq U$.

$X \twoheadrightarrow Y$: X-től többértékűen függ Y.

E előfordulás kielégíti:

$I \models X \twoheadrightarrow Y$, ha $t_1, t_2 \in I$ és $t_1[X] = t_2[X]$, akkor létezik $t_3 \in I$, hogy $t_3[X] = t_1[X]$, $t_3[Y] = t_1[Y]$ és $t_3[U - XY] = t_2[U - XY]$.

Összekapcsolási függőség:

$R(U)$ séma felett, $X_1, \dots, X_n \subseteq U$, $\bigcup_{i=1}^{n} X_i = U$.

jelölés: $\bowtie[X_1, \dots, X_n]$.

$$I \models \bowtie[X_1, \dots, X_n] \iff I = \Pi_{X_1}(I) \bowtie \dots \bowtie \Pi_{X_n}(I).$$

Implikáció a funkcionális, többértékű és összekapcsolási függőségekre eldönthető.

Többértékű és funkcionális függőségekre létezik helyes és teljes levezetési szabályrendszer.

(Összekapcsolásra nincs.)

Normalizálás: Lényege: egyszerű függőségi rendszerek teljesüljenek a relációkon. Lehetőleg minden kulcsfüggőségből következzen. Ezek a Normál Formák.

Módszer: Vetítéssel dekomponálunk, azzal egyszerűbb függőségi rendszer lesz a vetületeken. El kell kerülni az információvesztést:

- A vetületekből vissza kell tudni kapcsolni a kiindulási reláció előfordulását. (Veszteségmentes összekapcsolási dekompozíció: v.ö.d.)

Bemi

- Ha lehet, ne vesszünk függőséget: a vetületi függőségekből megkapható legyen az eredeti.

A v. ö. d. tulajdonságot minden dekomponálásnál megköveteljük. A függőségőrzés csak a 3NF esetében garantálható.

<u>Normál formák</u>: Az $R(U)$ séma, az $\mathcal{F}$ függőségi rendszerrel:

<u>Boyce-Codd Normálformában van</u> (BCNF), ha
$\mathcal{F} \models X \to A$, és $A \notin X$, $XA \neq U$, akkor
$\mathcal{F} \models X \to U$. ($X$ (szuper)kulcs.)

<u>3NF-ben van</u> (3. Normál formában van), ha
$\mathcal{F} \models X \to A$, és $A \notin X$, $XA \neq U$, akkor vagy $\mathcal{F} \models X \to U$, vagy A <u>elsődleges</u> attribútum.
A elsődleges, ha A egy $\mathcal{F}$-szerinti <u>minimális</u> kulcs eleme.

<u>4NF-ben van</u>, ha
$\mathcal{F} \models X \to\to Y$, és $Y \cap X = \emptyset$, $XY \neq U$, akkor
$\mathcal{F} \models X \to U$.

<u>5NF-ben van</u>: ha minden függőség (összekapcsolási is) egy kulcsfüggőségből következik.

(4)

Demi

<u>Normalizálás:</u>

Az $R(U)$ sémájú relációhoz az $\mathcal{F}$ függőségi rendszert adtuk meg.

Dekomponáljuk $X_1, \dots, X_n$ attribútumokra való vetítéssel:

Kapjuk az

$$R_i(X_i) = \Pi_{X_i}(R),$$ vetületeket az

$$\mathcal{F}_i = \Pi_{X_i}(\mathcal{F})$$ vetületi függőségekkel.

A dekomponálás a.c.a. veszteségmentes összekapcsolású, ha

$$\mathcal{F} \models \bowtie[X_1, X_2, \dots, X_n].$$

<u>BCNF-re dekomponálás</u>: Létezik v.ö.d. dekomponálás:

Lépésenként két részre bontunk egy relációt, amelyik megsérti a BCNF-et:

$S(V)$ sémájú reláció G függőségi rendszerrel megsérti BCNF-et ~~vagy~~ $G \models X \to A$ függőséggel, vegyük az

$$S_1(XA) = \Pi_{XA}(S) \quad \text{és} \quad S_2(V-A) = \Pi_{V-A}(S)$$

~~S_1 S_2~~ vetületeket.

Belátható egyszerűen, hogy

$$X \to A \models \bowtie[XA, V-A],$$

azaz S dekomponálása ~~a~~ G teljesülése esetén v. ö. d.

Ezt ismételjük, amíg minden BCNF-ben nem lesz. (A vetületeken a vetületi függőségi rendszereket kell használni!)

Függőségőrző dekomponálás általában BCNF-re nem lehetséges.

Bemi

3NF-re dekomponálás:

Létezik egyszerre v.ö.d. és függőségőrző dekomponálás.
(Algoritmus alapja: minimális függőségi rendszerét vesszük F-nek, ami $X_i \to A_i$ alakú függőségekből áll. Az X_iA_i halmazokra vetítünk, és még egy X kulcsra.)

4NF-re dekomponálás

Hasonlóan BCNF-hez, minden lépésben két részre bontunk, az $X \to\to Y$ függőséget használva XY, és $X \cup [U-Y]$ halmazokra vetítve.

5NF-re dekomponálás

Ha egy $\bowtie[X_1, .., X_n]$ függőség sérti meg az 5NF-et, vetíthetünk $X_1, .., X_n$ mindegyikére.
Egyébként a 4NF, vagy BCNF lépést hajthatjuk végre.

Bemi

Megszorítások és triggerek az SQL-ben

Irodalom: U-W.: 6. fejezet.

Az adatbázis működése az indokolt mértékű normalizált sémájú adatbázison történik.

Az adatbázis módosítása SQL-ben:
INSERT, DELETE, UPDATE utasítások.

A módosítások tranzakciókba szervződnek: műveletsorozat, teljes egészében végrehajtandó, vagy teljesen hatástalan. Végeredménye korrekt tartalmat jelent.
A korrektség (konzisztencia, épség) ellenőrzésére feltételek, megszorítások írhatók elő.
Az ABKR ezeket ellenőrzi, és betartja.

Az SQL-99-ben szabványosak szerepelnek.
A könyvben ezek még SQL3 hivatkozással szerepelnek.

Kulcsok, idegen kulcsok, hivatkozási épség

Első előadáson már szerepelt.
A CREATE TABLE utasításhoz tartozik:
- PRIMARY KEY (<attribútumok>) – több attribútum
- <attribútum deklaráció> PRIMARY KEY – egy attribútum a kulcs

Hivatkozás:
FOREIGN KEY <attribútumok> REFERENCES <táblanév> (<attribútumok>)

Mit ellenőriz az ABKR?
- A PRIMARY KEY esetén új sor beillesztésénél a kulcs egyedi voltát
- FOREIGN KEY esetén a hivatkozott táblában a kulcshoz tartozó sor létezését.

(7)

Bemi

A FOREIGN KEY hivatkozás egy attribútum esetén más alakban is megadható:

< a név > REFERENCES < táblanév > (< attribútum >) .

A hivatkozási épség ellenőrzését

ON DELETE ,

ON UPDATE

esetben tartozó lépéseket lehet még kiegészíteni. (nullára állítás, továbbgyűrűző módosítás)

Attribútumok értékeire vonatkozó megszorítások

A CREATE TABLE utasításon belül attribútumok deklarációjában használható lehetőségek:

- NOT NULL feltétel : nem vehet fel nullértéket

  < attribútumnév > ... NOT NULL

- CHECK feltétel attribútumra:

  < attribútum-deklaráció > CHECK (<u>név</u> Θ érték)

  a deklarált <u>név</u> attribútum értéke kielégíti a Θ szerinti feltételt. (Θ lehet: ≥, ≤, =, >, <, ≠)

  Másik lehetőség: értékek felsorolása:

  < attribútum-deklaráció > CHECK (<u>név</u> IN lista)

  A deklarált <u>név</u> attribútum értéke a listában szereplő valamelyik érték lehet.

- Értéktartomány definiálása is használata

  CREATE DOMAIN < értéknév > < típus >

  CHECK (VALUE < feltétel >)

  Hatása: az < értéknév > értéktípusként használható, kielégíti a CHECK után szereplő feltételt.

(8)

Bemi

Attribútum deklarációban használata:

<név> <értéknév>

Hatása nagyjából, mintha az <név> után közvetlenül használnánk a CHECK feltételt.
Előnyös, ha többször akarjuk használni, módosítható. (ALTER DOMAIN lehetőség)

Az értékekre és a hivatkozási épségre tett megszorításokat lehet azonnal, minden SQL utasítás után ellenőrizni, vagy elhalasztani a tranzakció végéig. (DEFERRED).

Globális megszorítások

- sorra vonatkozó CHECK feltételek: egyetlen reláció sorairól vonatkoznak
- önálló megszorítások: teljes relációk több sorára egyszerűbben vonatkozó feltételek.

Sorra vonatkozó CHECK feltétel

CREATE TABLE utasításban az attribútumdeklarációk, PRIMARY KEY, FOREIGN KEY deklarációk után adható meg

CHECK (feltétel-formula) alakban.

A feltétel-formula a WHERE-ben használható feltétel, azzal a megkötéssel, hogy az adott tábla egy sorának attribútumai közötti fennálló feltételt ellenőriz.

(9)

Bani

Önálló megszorítások – ASSERTION

Szemben a korábbi megszorításokkal, amelyeket egyes értékre, vagy egy sorra ellenőriz a rendszer, az önálló megszorításokat teljes táblákra ellenőrzi.

Az utasítás formája:

CREATE ASSERTION <név> CHECK (<feltétel>)

A <feltétel> megadásában használhatók SELECT utasítások, azok eredményére

- EXISTS vagy NOT EXISTS feltétel, ha van az eredmény,
- érték-összehasonlító feltétel, ha egyetlen értéket ad vissza a SELECT (pl. aggregáló kérdés).

Megszorítások elnevezése és módosítása

A NOT NULL, CHECK megszorításokhoz név adható meg, amit a feltétel megadása elé CONSTRAINT <feltételnév> beszúrásával tehetünk meg.

Ezek az ALTER TABLE utasításban

DROP CONSTRAINT <feltételnév>, vagy
ADD CONSTRAINT <feltételnév> <feltétel>

módon használhatjuk törlésre, új feltétel hozzáadására.

(10)

Bemi

Triggerek az SQL-99-ben

(U-W. 381- ; az SQL3 és SQL-99 ebben teljesen azonosak.)

A triggerek eseményvezérelt eljárások:

Esemény - Feltétel - Művelet szabályok.

(Ez már az aktív adatbázisok területe.)

Esemény: definiálni kell,
tipikus események: sor törlés | beszúrás | módosítás
tranzakció befejeződés

Feltétel: ha az esemény bekövetkezik, ellenőrizni kell a feltételt. Ha teljesül, végre kell hajtani a Műveletet.

Művelet: Egy megírt tranzakció-program lefuttatása.

Az SQL-99 triggerei:

1. A művelet végrehajtása lehet a kiváltó esemény előtt - utána - helyette.
2. Mit használhat a művelet: a kiváltó esemény által beszúrt, törölt vagy módosított sorok régi és új értékeit.
3. Ha az esemény módosítás, akkor megadható, milyen oszlopra, vagy oszlopok halmazára vonatkozik az esemény.
4. Egy WHEN záradékban adható meg a feltétel.

11

Bemi

Tranzakciókezelés

Irodalom: Garcia-Molina, Ullman, Widom: Adatbázisrendszerek megvalósítása

A megszorítások, triggerek átvezetnek a tranzakció-kezelés, konkurenciakezelés területére.

A könyv kijezetei: 9. és 10.

Tranzakció: egy alkalmazási feladat (program, ügyintézés, ..) végrehajtása adatbázison.

Általános elvárás – ACID tulajdonság:

A – atomicity – atomosság

C – consistency – konzisztencia, épség, korrektség (pl. TRIGGER-ek szerepe)

I – izolation – elkülönültség, izoláltság

D – durability – tartósság

Tranzakció az ABKR szemszögéből:
az adatbázist érintő lépések sorozata. (Beleértve például a triggerek meghívásával keletkező lépéseket is.)

Legfontosabb lépések: olvasás és írás. Amit érint egy lépés: adatbázis elem. (A lépést atomosnak, felbonthatatlannak tekintjük.)

Szokásos jelölések a lépésekre: T: tranzakció azonosító
A, B, ..: Adatbázis elem
t: program változója

A T tranzakció lépései:

T: Read (A, t) – A beolvasása t-be

T: Write (A, t) – A kiírása (új értéke) t-ből.

Az ABKR tipikusan puffereket (puffer-blokkokat) használ. Ezeken keresztül történik a háttértároló lapjainak olvasása és írása.

Bemi

Az ABKR tipikusan még két további lépést használ:

- a Read(A,t) lépés előtt (valamikor):

  INPUT(A) : háttértárolóról a pufferbe olvasás.

- a Write(A,t) lépés után (valamikor):

  OUTPUT(A) : pufferből a háttértárolóra ír.

Az INPUT és OUTPU lépések a puffer-gazdálkodástól függnek, elválnak a tranzakciók Read és Write lépéseitől!

<u>Ütemezés</u> A tranzakciók lépéseit az ABKR összefésüli, konkurens módon, "párhuzamosan" hajtja végre.

<u>Definíció</u> A $T_1, T_2, \ldots, T_n$ tranzakciók <u>S ütemezése</u> az egyes tranzakciók lépéseinek <u>összefésülése</u>.
(összefésülés: az egyes tranzakciókra nézve lépéseik sorrendje nem kavarodik össze.)

<u>Ütemezések ekvivalenciája</u>

- <u>Végállapot ekvivalencia</u>: a $T_1, \ldots, T_n$ tranzakciók S és S' ütemezése <u>végállapot-ekvivalens</u>, ha ugyanabból az adatbázis-állapotból kiindulva ugyanazt a végállapotot hozzák létre. (Más néven: <u>nézet-ekvivalencia</u>)
- <u>Olvasási ekvivalencia</u>: ha a végállapot-ekvivalencián felül még minden T_i:Read(A_j, t_i) lépés mindkét ütemezésben ugyanazt az értéket olvassa.

(2)

<u>Konfliktus-szituációk</u> az S ütemezés lépései között:

(i) ..., T_i : Write(A); ..., T_j : Read(A); ...

és a két lépés között nincs A-ra írási lépés. Ekkor az a két lépés nem cserélhető fel. A T_j az A értékét közvetlenül a T_i -től olvassa.
Ekvivalens soros ütemezésben T_i meg kell, hogy előzze T_j-t.

(ii) ..., T_i : Read(A); ..., T_j : Write(A); ...

Olvasási ekvivalencia esetén nem cserélhető fel a két lépés. T_i mást olvashat, ha a T_j : Write(A) lépése megelőzi T_i : Read(A)-t.

<u>S átrendezhetősége soros ütemezésre:</u>

Az S ütemezés lépéseit megpróbáljuk felcserélgetni, úgy, hogy például T_i lépései megelőzzék az összes többi tranzakció lépését. Ha úgy sikerül átrendezni soros ütemezésre, akkor S <u>konfliktus-sorolható</u>.

<u>Hatékony ellenőrzés</u>: <u>sorolhatósági gráf</u> alapján.

A $T_1, \dots, T_n$ tranzakciók S ütemezésének G sorolhatósági gráfja: Csúcspontok: $T_1, \dots, T_n$
Élek: $T_i \to T_j$, ha T_i és T_j között valamilyen A szerint konfliktus-szituáció áll fenn. (Kényszerítő sorrendet jelent!)

<u>Tétel</u> S akkor és csak akkor konfliktussorolható, ha G körmentes.

③

Bemi

<u>Soros (szeriális) ütemezés</u>: A $T_1, \dots, T_n$ tranzakciók V ütemezése soros ütemezés, ha a tranzakciók egymás után, elkülönülten valamilyen $T_{i_1}, T_{i_2}, \dots, T_{i_n}$ sorrendben hajtódnak végre.
(Ezt nyilván helyes végeredményre vezető végrehajtásnak tekintjük. I és C elvárások rendben vannak.)

<u>Sorolhatóság</u>
Def. Az S ütemezés <u>sorolható</u>, ha létezik vele <u>ekvivalens V soros</u> ütemezés.

Végállapot ekvivalencia esetén | nézet <u>végállapot-sorolhatóságról</u>, Olvasási ekvivalencia esetén <u>olvasási-ekvivalenciáról</u> beszélünk.

A sorolhatóság ellenőrzése általános esetben nehéz (NP-teljes), ezért egy korlátozottabb (szigorúbb feltétel melletti) sorolhatóságot szokás ellenőrizni: a <u>konfliktus-sorolhatóságot</u>.
(A konfliktus-sorolhatóságból következik az olvasási és a végállapot szerinti sorolhatóság.)

A konfliktus-szituációk egy ütemezésben az azonos adatbázis-elemre vonatkozó írási és olvasási műveletek felcserélhetetlenségét jelentik.

<u>Először feltesszük</u>, hogy a T: Write(A, t) lépés az A előző értékét felhasználja, másként, a T beolvassa az írás előtt A értékét.

(4)

Bennő

(Csak (i) típus figyelése: végállapot-sorolhatóság.
Mindkét típus – (i) és (ii) figyelembe vétele: olvasási-sorolhatóság.)

<u>Még finomabb modell</u>: GM-U-W: 10.2. szakasz: „Nézetsorbarendezés"

Új feltevés – új lehetőség: a T: Write(A, t) lépést megelőzően a T tranzakció csak akkor kapja meg A megelőző értékét, ha megelőzően explicit módon T: Read(A, t') lépést hajt végre.

Az általános nézet-ekvivalencia ebben az esetben még nehezebben ellenőrizhető.
~~Ez~~ Ezért az (i) és (ii) konfliktusok ~~helyett~~ mellett még egy konfliktus-szituációt veszünk figyelembe:

(iii) Ha van az S ütemezésben

T_i : Write(A); ..., T_j : Read(A); <u>közvetlen</u>

olvasási (vagyis a két lépés között nincs A-ra írási lépés), és van valahol még egy

T_k : Write(A) lépés, akkor T_k vagy

T_i előtt, vagy T_j után lehet.

Ennek megfelelően a sorolhatósági (megelőzési) gráfban

alakú <u>poliél</u> keletkezik. A raggatott élek közül az egyiket kell választani.

(5)

Ebben az esetben a sorolhatósági gráf, G irányítatlan poli-gráf lesz.

<u>Tétel</u> Az S ütemezés az (i), (ii) és (iii) konfliktusok szerint konfliktus-sorolható akkor és csak akkor, ha a poliélekre van olyan választása, amelyen a keletkező közönséges gráf körmentes.

— — — — —

A tranzakciók olvasási és írási lépéseinek megfigyelésére alapuló sorolhatósági ellenőrzés túl sok hibás működést észlelne. A hibát okozó tranzakciót ilyenkor hatástalanítani kell.

A konkurens használata ugyanannak az elemnek jobban korlátozható, ami megelőzhet hibás lépéseket.

Erre a célra vezették be a
- zárak (lock-ok) használatát, vagy az
- időbélyegzők használatát.

<u>Zárak</u> Olvasás, írás előtt hozzáférési engedélyt kell szerezni az elemre. Az engedély zárolja az elemet további hozzáférés elől az engedély feloldásáig. Az ütemezőnek elég csak a zárakra vonatkozó lépéseket ütemeznie. A zárak engedélyezését a zárkezelő (lock-manager) végzi.

Tranzakciók (ütemezendő) lépései:

T_i : X-Lock(A) ; ahol X : a zár típusa,

T_i : UnLock(A) ; a zár elengedése. Minden A-ra vonatkozó zárról lemond a T_i tranzakció.

6

Bunni

A könyv saját jelölést használ:

Lock - típus	könyv jelölése	
T_i : ReadLock(A)	$rl_i(A)$	- írási zár
T_i : WriteLock(A)	$wl_i(A)$	- olvasási zár
T_i : UpdateLock(A)	$ul_i(A)$	- módosítási zár
T_i : SharedLock(A)	$sl_i(A)$	- osztott zár
T_i : ExclusiveLock(A)	$xl_i(A)$	- kizárólagos zár
T_i : UnLock(A)	$u_i(A)$	- zár elengedése

Zár engedélyezése: a zár-kezelő feladata.
a zár-konfliktus tábla alapján engedélyez.

Például sl és xl típusok esetén:

Foglalt / Igényelt	sl	xl
sl	Igen	Nem
xl	Nem	Nem

Ütemezés zárak esetén: csak a zárási és zárelengedési lépéseket figyeli és ütemezi.

Jogszerű (legális) ütemezés: a zárkezelő engedélyezett minden ütemezett zárigényt.

<u>Kétfázisú zárolási protokoll:</u>

A tranzakció az első zárelengedés (unlock lépés) után más nem igényel több zárat. (7)

Bmi

Tétel Kétfázisú tranzakciók bármely jogszerű ütemezése sorolható.
(Az **első** **unlock** lépések sorrendje megfelelő soros ütemezést ad.)
Bizonyítás: A konfliktus-szituációkat zárakra értelmezzük. A sorolhatósági gráf körmentes lesz.

Meghibásodás elleni védelem

Hibák lé típusai:

- **Tranzakció hibás** lépést hajtana végre. (aritmetikai hiba, jogosulatlan hozzáférés stb.)
- **ütemező**, (lockmanager) zárkezelő észlel sorolhatósági vagy patthelyzet hibát és **abortálja** a hibázó tranzakciót
- **Rendszerhiba**: gép leáll, operációs rendszer leáll; elvész a memória, pufferek tartalma
- **Háttértároló meghibásodás**.

Megoldási technikák

Az **abort** lehetősége miatt a tranzakciók további két lépést jelznek: T : BEGIN - a kezdés,
T : COMMIT - jó befejezés
T : ABORT - érvénytelen befejezés.

Napló (log-journal-) fájl készítése:

szekvenciálisan, ⟨T : BEGIN⟩, ⟨T : COMMIT⟩, ⟨T : ABORT⟩, ⟨T : A, újérték /és, vagy/ régi érték⟩ (8)

elem

bejegyzések kerülnek a naplóba. A napló blokkjai folyamatosan kiírásra kerülnek a háttértárolóra.

<u>Visszaállító napló</u> (ROLLBACK): az A elem régi értéke kerül a naplóba. ABORT esetén visszatölti az adatbázisba a régi értéket.
(Piszkos adat olvasása: abortált tranzakció által írt adat olvasása. Továbbterjedő ROLLBACK)

<u>Újrakészítő napló</u> (REDO): Az A elem új értéke kerül a naplóba. A kétfázisú zárási protokoll rugalmasságával együtt használható.
<u>Rendszerhiba</u> kivédésére szolgál. A napló elejéről a COMMIT-tal végződött tranzakciók új-értékeit a napló elejétől kezdve visszaírja az adatbázisba.

A naplót időközönként le kell zárni, és újat kell kezdeni. Ez a <u>kontrollpont-készítés</u>.

<u>Háttértároló</u> hibák kivédése: alapvetően csak <u>többszörözéssel</u> lehet.

- Biztonsági másolatok készítése.
  (kiegészíthető a REDO naplójával)
- Tükrözés
- RAID rendszerek, egy, vagy két tengelyen keletkező hiba kijavítása.
  Olvasásra kihasználható a többszörözés, írásnál többletmunkát igényel.

(További technikák: osztott adatbázisok, hierarchikus elemek, több mélységű tranzakciók)

9

Bui

Az objektum-relációs modell
(Object-Relational Model)

/ U-W könyv 8.5. és 8.6. szakaszok. /

Az objektum-orientált adatbázis-kezelők (ODL, OODBMS) nem tudták kiszorítani, lecserélni a relációs rendszereket.

Az következett be, hogy a relációs rendszerek szállítói több objektum-orientált elemet építettek be a relációs rendszerekbe.

Relációktól az Objektum-Relációkig

Új elemek a modellben:

1.) Az attribútumokhoz összetett, szerkezettel rendelkező típusú adatok is rendelhetők. Szemben a korábbi atomi értékekkel.
A tipikus konstruktorok: halmaz, multihalmaz, sor.
Például: sorokból álló halmaz is lehet egy attribútum értéke, vagyis egy reláció.

2. Metódusok. az objektum-orientált modellek mintájára.

3. Azonosító rendelhető sorokhoz. (Más, mint a kulcs!)
A sor tölti be az azonosítóval rendelkező objektum szerepét. Az általános ODL-hez visszanyúlva ez azt jelenti, hogy itt az objektumokhoz tartozó adat-típus csak sor lehet.

4. Hivatkozások (References) A sor-azonosítókra lehet hivatkozni, pointerként használható; többféle módon.

Bui

Beágyazott relációk (Nested Relations)

Speciális eset. Már a 80-as években ismert volt.

A szerkezet képzésében csak alternálva használható a sor - halmaz konstrukció. Ez azt jelenti, hogy egy attribútumhoz értéktípusként relációséma rendelhető, és érték gyanánt pedig relációelőfordulás (tábla).

A séma építésének lehetőség formálisan:

Alap: Egy atomi típus (egész, logikai, sztring, ..), ami egy attribútum típusa lehet.

Indukció: Egy reláció típusa lehet bármilyen séma, ami egy vagy több attribútumnévből áll, és minden attribútum típusa legális. Az attribútum típusa is lehet ilyen séma.

Az első előadás jelölését folytatva egy séma:

R(⟨A : egész, B : ⟨C : sztring, D : sztring, E : ⟨F : egész⟩⟩⟩)

(11)

Bmi

Például:

Hallgató(< név: string, lakcím: (város: string, utca: string, házsz: egész),
szülidő: dátum, tantárgyak: (azonosító: egész, tnév: string,
év: dátum, jegy: egész>>)

Ebben két beágyazott reláció van hallgatónként.

Más példák: családok; filmek és filmszínészek adatbázisa

Hivatkozások, References

A beágyazott relációknál előfordulhat, hogy egy sor több beágyazott táblában is szerepel.

Például: filmek adatai egy színész adatsorában beágyazva.

Ha egy ilyen sornak van azonosítója, akkor elég egyszer tárolni, és csak rámutató pointert, hivatkozást használni.

Jelölési konvenció adatmodellezésben:

- az R nevű reláció egy sorára történő hivatkozás: *R

- az R nevű reláció több sorára történő hivatkozás: ~~{*R}~~ {*R}.

A : *R
A : {*R}

eddigi jelöléseink szerint: az A attribútum értéke: hivatkozás.

Objektum-orientáltság az objektum-relációs modellben (ORM)

(1) Az ORM-ben a sor töltheti be objektum szerepét, és csak attribútumaihoz rendelhető összetett érték.

Feloldja az Egyed-Kapcsolat modell reprezentálásában a relációs modell nehézkességét:

(III)

Bmi

Nem kell külön relációban reprezentálnia a kapcsolatokat, hanem használhatók a hivatkozások. Tehát, ha objektumoknak (entitásoknak) sorokat feleltetünk meg, a sorok azonosítói használhatók az objektumok azonosítójaként.

(2) Az SQL-99 is az ORM más implementációjú lehetővé teszik metódusok definiálását az egyes típusokhoz.

(3) Adattípus: hasonló lehetőség, mint az OODB-ben.
építhető: atomi típus, sor-típus (struktúra), halmaz, vagy multihalmaz.
Mind az ORM-ben, mind az OODB modellben kiemelt fontosságú a sorokból képzett halmaz.

(4) A hivatkozás, mint objektumazonosító (OID)
A tipikus OODB-ben az objektumazonosító teljes mértékben hozzáférhetetlen.
Az ORM-ben a hivatkozás elérhető típus, így látható, megjegyezhető.
Elvben ez különbség; gyakorlati rendszerben minimális szerepet játszik.

Az ORM rendszerek visszafele (a tiszta relációs régebbi verziókig) kompatibilisek. Ezért nyertek a tiszta OODB-kel szemben.

(IV)

Bui

Felhasználó által definiált típusok az SQL-99-ben

User-Defined Types (UDT)

Ez az alapja az ORM lehetőségek beépítésének.

UDT lehet: (tábla ≡ reláció)

1. Egy tábla típusa
2. Egy attribútum típusa, ami valamilyen táblához tartozik.

Egyszerű eset: alaptípus átnevezése

pl.

```
CREATE TYPE Höjdh AS INTEGER
CREATE TYPE Mosz AS INTEGER
```

Összetett eset:

Tábla-típus használata UDT-ként.
A relációkon megadható kulcs, függőség, ettől független. Az nem tartozik a típushoz!

Alakja: CREATE TYPE T AS (<attribútum deklarációk>)

Pl:

```
CREATE TYPE Lakcímtípus AS(
        utca CHAR(50),
        város CHAR(20)
);

CREATE TYPE Személytípus AS(
        unév CHAR(20),
        vnév CHAR(20),
        lakcím Lakcímtípus
);
```

(V)

Bui

Metódus deklaráció UDT-hez:

Az ~~UD~~ <u>UDT deklarációban kapcsolva:</u>

CREATE TYPE (típusnév) AS (<attribútumdeklarációk>)
METHOD (metódusnév) (<argumentumdeklarációk>)
RETURNS <eredménytípus>

A metódus törzsét külön kell megadnia (pl. JAVA).
A metódus a típusban deklarált sorra vonatkozik.
SELF – a hívás kurzus sorára hivatkozik
Lehet üres az argumentumlista.

Pl:
```
CREATE TYPE Lakcímtípus AS (
        utca CHAR(50),
        város CHAR(20)          )
    METHOD házszám() RETURNS CHAR(10);
```

<u>A metódus elkülönült definiálása:</u>

```
CREATE METHOD <metódusnév, argumentumok,
        visszatérési típus>
        FOR <UDT név>
            <metódus törzs>;
```

Pl:
```
CREATE METHOD házszám() RETURNS CHAR(10)
    FOR Lakcímtípus
    BEGIN
        ... (Tetszőleges string, ami a program kódja.)
    END;
```

<u>UDT használata relációdeklarációban</u>

```
CREATE TABLE <relációnév> OF <UDT név>
    (<speciális elemek listája>);
```

A speciális elemek: PRIMARY KEY,
FOREGN KEY,
sorra vonatkozó feltételek, .

(VI)

Bemí

Hivatkozások (References)

Táblák sorairól másik tábla attribútuma hivatkozhat.
A T felhasználói adattípus (UDT) hivatkozási oszlopa mindig REF(T).
Egy ilyen típusú sorra való hivatkozásnál megadható az is, melyik reláció sora, ez a scope.

Például: A REF(T) SCOPE R

jelentése egy A attribútum definiálásánál:
A egy T típusú sorra hivatkozik az R relációból.
Ha nincs SCOPE, akkor bármely T típusú sorra mutathat a hivatkozás.

Sorazonosítók letrehozása Táblákhoz

Az UDT-vel deklarált táblákhoz a speciális elemek közöt definiálható hivatkozás.
Az ilyen táblák hivatkozhatók (referenceable):

REF IS <attribútumnév> <generálás módja>

Generálás módja:

1. SYSTEM GENERATED, az ABKR rendel egyedi értéket minden sorhoz ebben az attribútumban.
2. DERIVED, ilyenkor a tábla elsődleges kulcsát használja egyedi érték meghatározásához.

VII

Bem

Példa Filmek, Szereplők, Filmszínészek (U-W új kiadás, 2008.)

```
1)  CREATE TYPE Filmtipus AS (
2)          cim     CHAR(30),
3)          év      INTEGER
4)          tipus   CHAR(10)
            );
```

játékfilm, krimi, rajzfilm ..

```
5)  CREATE TABLE Filmek OF Filmtipus(
6)      REF IS filmID SYSTEM GENERATED
7)      PRIMARY KEY ( cim, év)
        );
```

— — — — — —

```
CREATE szinésztipus AS (
        név     CHAR(30)
        lakcím  Lakcímtipus
        legjobbFilm REF(Filmtipus) SCOPE Filmek
        );

CREATE TABLE Szinész AS Szinésztipus (
    REF IS szinészID SYSTEM GENERATED
    PRIMARY KEY (név)
    );

CREATE TABLE Szereplője (
    szinész REF(Szinésztipus) SCOPE Szinész,
    film    REF(Filmtipus) SCOPE Filmek
    );
```

(VIII)

Bemi

Műveletek az objektum-relációs adatokon

Általában használhatók a közönséges relációhoz értelmezett műveletek. Két teljesen új művelet kell a hivatkozás-követésre, és az UDT típusú értéket tartalmazó oszlop kezeléséhez.

Hivatkozások követése

Legyen x egy REF(T) típusú érték.
Akkor x egy T típusú t sorra hivatkozik.

1. A t sor A attribútumára
   x → A hivatkozik.
2. A teljes t sorra a DEREF operátorral hivatkozhatunk.

Példa: a szereplője (színész, film) reláció:

```
1)  SELECT  DEREF(film)
2)  FROM    szereplője
3)  WHERE   színész → név = "XY";
```

(A színész és a film hivatkozás-típus!)

(Az egyszerű SELECT film
FROM szereplője, ..
: ez csak a gépi által generált azonosítókat adja vissza!)

UDT típusú komponensek kezelése

Egy UDT-vel adott tábla sortípust jelent, a teljes sort egyetlen objektumként kezelhetjük.

A komponensekhez lefúrással (drill down) juthatunk.

Ezt az UDT alapértelmezésű megfigyelő metódusa biztosítja (observer method)

Az x attribútum megfigyelő metódusa x().

(IX)

Bemi

A t sor, T típusú, x attribútumára
t.x() jelöléssel hívjuk a megfigyelő metódust.

Pl:
```
SELECT f.év()
FROM Filmek f
WHERE f.cím() = "KingKong";
```

Gyakorlatban a pontos metódushívás egyszerűsíthető, a () zárójelpár elhagyható.

```
SELECT f.év
  FROM Filmek f
  WHERE f.cím = "KingKong";
```

A megfigyelő metódus több mélységben használható:
s.lakcím.házszám().

UDT típusú adat létrehozása és komponenseinek módosítása:

- Generátor metódus (Generator method):
  T típus, T() : egy üres komponensekből álló példányának generálása

- Változtató metódus (Mutator methods)
  Hasonlít a megfigyelő metódushoz: az x attribútum értéke legyen v : x(v)
  A t.x(v) jelentése: a t sor x attribútuma vegye fel a v értéket.

Ezt a két metódust implementálják az O-R adatbázisrendszerek.

ODT-k rendezése (kellhet több SQL utasításhoz)

1. Típus: CREATE ORDERING FOR T EQUALS ONLY BY STATE;
   (teljes tartalom egyezés)

2. Típus: CREATE ORDERING FOR T
   ORDERING FULL BY RELATIVE WITH F;
   (Az $F(x_1, x_2)$ függvényt kell megadni!)

Bmi

Fájlkezelés

Irodalom: GM-H-W: Adatbázisrendszerek megvalósítása
3. kiadat. 111-152 old.

Alapfeladata: rekordok elhelyezése és visszakeresése háttértároló blokkjaiban/blokkjaiból.

Rekordszerkezetek: mezőkből épülő hierarchikus szerkezet. Rekord fejléce.
Kulcsmező: értéke egyedi a fájl rekordjain.

Fájl: háttértároló blokkok csoportja, a fájlhoz tartozó rekordok elhelyezésére szolgál.

A blokkok **címezhetők** (hardver cím, logikai cím)

Fájlrendszer – **fizikai adatmodell**, a tárolás szintjének adatmodellje.

Az ABKR-ek a normál felhasználók elől ezt elrejtik.

Konkrét feladatok:
- **Blokkon belül**: mi van a blokkban (kis), rekordok eleje, vége, ...
- **Blokkok között**:
  - átnyúló rekordok folytatása,
  - pointerek, egymásra mutató rekordok,
  - blokkláncok, ...
  - Fájl fejrekord szerepe...

- **Rekordok elhelyezése**: milyen módon rendeljük a rekordokat blokkokhoz, hogyan tesszük lehetővé visszakeresésüket

(1)

Bmi

Bemi

- Memória puffer-blokkok és háttértároló blokkok közötti mozgatás.
  Virtuális blokkcím: a pufferben levő blokkra mutat
  Fizikai blokkcím: a háttértároló címzési módja szerint rögzített cím.

<u>Blokkcímek</u>, <u>rekordcímek</u> ábrázolása

Fizikai Rekordcím = (blokkcím, blokkon belüli eltolási érték)

Logikai cím: memórián belüli aktuális cím; fordítási tábla kezeli

Rekordra (rekordcímre) mutató pointerek kezelése.

<u>Változó hosszú</u> rekordok kezelési módszerei -
alaprekord - folytató, túlcsordulási rekord, --

<u>Elhelyezés - visszakeresés - indexek - hasító táblák</u>

<u>Irodalom</u>: GM-U-W: 4. kiadat. 153-214.

/ Relációs rendszerek: tábla sora ↔ rekord
fájl: egy, vagy több tábla sorait tárolja /

<u>Mit hol helyezünk el</u> - rekordazonosító, blokkcím megfeleltetés.

<u>Alapmegoldások:</u>

- nincs megfeleltetés: visszakereséshez minden blokkot meg kell nézni. (Nagy blokkhalmaz)

- táblázatban kezeljük a rekordazonosító - blokkcím párokat. (Indexfájlok)

- számítással adjuk meg, mit hol tárolunk (hasító táblák)

(11)

Bri

Elsődleges index: rekordazonosító (kulcs) értéke alapján gyors visszakeresés

Sűrű index: minden rekordhoz a (rekord-azonosító, ~~blokk~~ rekordcím) párok feljegyzése.
(Ez is egy fájl, lényegesen kisebb, mint az alapfájl)

Ritka index: minden blokkhoz csak egy indexérték: a blokk első (legkisebb) rekordjának azonosítójához.
Feltétele: a blokkokban lévő ~~(legnagyobb, t~~ (legkisebb, legnagyobb) rekordazonosító értékpárok ~~diszj~~ diszjunkt intervallumokat alkotnak.

Lehet az alapelhelyezés rendezett - nem rendezett. (Indexszekvenciális fájlok)

Az indexfájl mindkét esetben rendezett.
Beillesztés - törlés kérdései!

Többszintű index elsődleges indexekre

Az első indexfájlra lehet ritka indexet készíteni.
Az így kapott indexfájlra megint lehet, ...
Általában a legfelső index már a memóriában tartható.

Bui

Másodlagos indexek, index ismétlődő értékre

Másodlagos index: csak sűrű lehet

Ismétlődő értékre: értékenként rekordcímek listája, ez változó hosszú rekord, a fejben indextábla, a fej-rekordhoz felfűzzük a címlistát. (Kosarak)

Indexfájlok kezelési kérdései: keresés, bezúrás – túlcsordulások

Többszintű indexfájl – megjelul egy keresőfának.

B – fák

Legelterjedtebb indexszerkezet.

Keresőfa: rendezhető értékek alapján keres. A fában balra haladva kisebb, jobbra nagyobb keresési értékeket tartalmazó ágak találhető.

Bináris keresőfa: a fokszám legfeljebb 2.

1. ábra: belső pont is kulcsértékek

2. ábra: csak levélben van kulcsértékek, Belső kulcs – nem mind szerepel levélben is!

(IV)

Bui

A B-fák olyan keresőfák, amelyekben egy k paraméntől függően minden belső pontnak legalább k+1 gyereke, és legfeljebb 2k+1 gyereke lehet. Kivétel a gyökér, amelyre az alsó határt 2.

Az eredeti B-fában a belső keresési értékek is ~~lehet~~ kulcsértékek, rekordcímmel.
A B^+-fában a levelekben vannak a kulcs, rekordcím párok.

A B- és B^+ fák kiegyensúlyozottak.

A fa magassága n kereshető kulcsra $\log_{k+1} n$ és $\log_{2k+1} n$ közé esik.

A fa pontjait általában egy-egy blokk tartalmazza. 4 Kbyte méretű blokkokra k értéke tipikusan 100 nagyságrendű.

Belső pontnak megfelelő blokk felépítése j+1 gyerekkel (mutatóval): m_i : az i-edik mutató,
K_i : az i-edik kulcs,
$K_1 < K_2 < \dots < K_j$

Bi

Kulcs beillesztése: - melyik levélhez tartozik
- ha van szabad hely, beillesztődik a blokkba
- ha nincs: 2k+2 kulcs lesz benne, osztódik, és a bán felfele beillesztése az új pontnak.

Kulcs törlése: - ha k+1-nél több a fekszám, törölhető egy keresési kulcs és mutató.
- ha éppen k+1 ~~keresi~~ mutató van, meg kell nézni a két szomszédját:
  - ha valamelyikben k+1-nél több mutató van, egy mutatót áthozunk a törlés után.
  - egyébként összevonjuk a testvérrel, és 2k+1 mutató marad, a csomópont (blokk) törlődik, ez felfele új törlés lesz.

Keresés, tartomány-keresés

adott kulcsértékre — intervallum adott a keresett kulcsokra

Alkalmazások: a korábban említett index fájlok mindegyikére építhető B^+-fa.

A B-fa eredete: Bayer és McCreight 1972.

B: - utal a Balanced (kiegyensúlyozott) tulajdonságra,
- lehet, hogy Bayer nevére utal.

Tördelő táblázatok (Hasító táblázatok)

Kiindulás: az rekord elhelyezési helyét a rekord kulcsából számítással határozzuk meg.

Az r rekord elhelyezéséhez k kulcsából egy h tördelő függvénnyel egy $h(k) = c$ értéket számolunk ki.

A tördelő táblázatos elhelyezésnél az a c érték egy blokkhalmazból álló kosár kezdő blokkjának címe, A (c, blokkcím) párokból álló táblázatot általában a memóriában tároljuk.

Keresés – beillesztés, túlcsordulás – törlés megoldásai.

Kiterjeszthető (dinamikus) tördelőtáblázatok:
telitett kosarak kettő osztása – egy bittel hosszabb tördelő érték.

Többdimenziós indexek – tartománykeresések

Irodalom: GM-U-W: 5. fejezet, 216–263.

Tördelésen alapuló módszerek:
particionált tördelés
rács-indexelés, hasító táblával kiegészítve

Fa-struktúrák: (Térinformatika)
- többkulcsos indexek
- kd-fák
- negyedelő, nyolcadoló fák (Z-bejárás)
- R-fák (téglalapokból épül, Rectangle)

Bittérképek

Particionált tördelés

A rekord több mezőjére egyszerre történő keresés.

A mezők: $F_1, \dots, F_k$.

Mindegyikhez egy-egy tördelő függvényt rendelünk: $h_1, h_2, \dots, h_k$.

Az egyes függvények $l_1, \dots, l_k$ hosszú bitekből álló bináris számokat állítanak elő.

Összefűzésükkel $l = \sum_{i=1}^{k} l_i$ hosszú bináris számot kapunk. Ezt használjuk a kosár táblájának címzésére.

Például $F_1 = \alpha_1, \dots, F_k = \alpha_k$ értékekre a

$c_1 = h_1(\alpha_1), \dots, c_k = h_k(\alpha_k)$ bináris számokból

a $c = c_1 c_2 \dots c_k$ tördelő címet kapjuk.

Részleges keresést is támogatja.

Például, ha F_i értékét nem adjuk meg, csak a többit, akkor a megfelelő helyen l_i bit hiányzik, ezért 2^{l_i} különböző értékre kell megnézni az l hosszú lehetséges címeket.

Rácsos állományok – Rácsindexek

Két mezőre mutatjuk be: F_1 – $[X_1, X_2]$

F_2 – $[Y_1, Y_2]$

A tartományt felosztjuk cellákra, a cellákhoz rendelünk kosarat.

Az (x,y) pont a $[i,j]$ cellába esik, $[i,j]$ kosarában tároljuk.

(VIII)

Bui

Fa-struktúrák

Többkulcsos index: F_1-re, F_2-re, ..., F_k-ra egymást követő indexfa.

Az F_1 szerinti keresés leveléhez egy F_2 szerinti kereső fát építünk F_2 értékeire, stb.

(Tetszőleges típusú mezőkhöz használható.)

A kd-fák

A fa egyes szintjein más-más mező szerint történik a csomópont alatti rekordok szétosztása, (particionálása) a keresőfák elve szerint. (rendezett sz.)

$F_1, \dots, F_k$ mezők esetén először az F_1-re, aztán F_2-re, ..., F_k-ra, majd újra F_1-re, ------, —"— ...

A rekordok / rekordcímek a leveleken lesznek.

(Mező típusa tetszőleges, rendezett érték)

Negyedelő fák (Quad-fák síkon, nyolcadoló-fák térben)

Negyedelés: A negyedeket szögletes bejárással azonosítjuk:

Ezt folytatjuk:

Fa:

Bmi

Pontok indexelése negyedelő fával

Felbontjuk a tartományt a pontok eloszlása alapján kvadránsokra. Jó, ha közel azonos számú pont esik minden kvadránsba. (A könyvben egy vagy kettő.)
A kvadrán azonosítóját használjuk a keresőfa felépítéséhez.

(Térinformatikai használat:
Tartományt indexelünk lefedő kvadránsaival.
a.) Egy lefedésben szereplő kvadránst nem bontunk tovább, ha a tartomány mind a 4 kvadránsába belemetsz.
b.) Másik változat: rögzített kvadránsmélységig adjuk meg a fedő kvadránsokat.

Pl. A t-t lefedő kvadránsok:
1, 2, 300, 301, 302

t pontjainak indexelését a (t,1), (t,2), (t,300), (t,301), (t,302) „rekordok" keresőnegyedelő-fába illesztésével kapjuk.
/ (t,q) párok /

Z-rendezés : kvadránsokra

Tetszőleges kvadránsfelbontás egyetlen Z-vonallal bejárható.

A Z-rendezés szerint a kvadránsok B^+-fával indexelhetők. A (t,q) párokat ezzel B^+-fával indexelhetjük.)

Bmi

(X)

Beni

R - fák (Régió-fák, Rectangle-tree)

Tartományokat megadó adatok indexelése.
Térinformatikában használják.
Térbeli objektum geometriai adata: tartomány.
A tartományhoz minimális befoglaló téglalapot rendelünk:

t:

A téglalapokra építünk keresőfát.
A keresőfában a belső csomópontokhoz rendelt téglalap a gyerekeihez tartozó téglalapokat lefedő legkisebb téglalap.
A B-fákhoz hasonlóan a gyerekek száma $k+1$ és $2k+1$ között lehet, rögzített k paraméterrel.

Egy csomópont a fában általában egy háttértároló blokk méretű.
Felépítése: a t, $t_1, \dots, t_n$ téglalapokból áll,
ahol t : a csomópont téglalapja,
$t_1, \dots, t_n$: a gyerekei téglalapja.

A levelekben: a tér-objektumok téglalapjai és tárolási címei szerepelnek.

XI

BI

Bittérképindexek

Legyen n rekordunk (sorunk), $1, \dots, n$ -ig sorszámozva.
Az A mező (attribútum) "a" értékéhez n bitből álló vektort rendelhetünk: az i-edik bit legyen 1, ha az i-edik rekordban A értéke "a", egyébként legyen 0.

Az A mező lehetséges $a_1, \dots, a_m$ értékeihez elkészítjük az m bitvektort.
Ez az A mező bittérképe. Mérete: mn bit.

Több mezőre felépítve többdimenziós keresést segíti.
Pl: az $A =$ "a" ÉS $B =$ "b" feltételre keresés az "a"-hoz és a "b"-hez tartozó bitvektorok logikai AND műveletével megadja a keresett rekordok sorszámát.

Kezelésének részletei:

- Ha m nagy, akkor a bitvektorok ritkák, kevés 1-es lesz bennük: <u>tömörítés</u>

- Rekordok változása, beillesztése, törlése; átvezetés a bittérképeken

XII

BS

Bai

Lekérdezések végrehajtása

- Kérdés (SQL) elemzése: elemző fa építése (algebrai operátorokkal)
- Átalakítás – ekvivalens kifejezésre, optimalizálási lehetőségek szerint
- Fizikai végrehajtási terv:
  - műveletek sorrendje,
  - adatátadás módja műveletek között
  - háttértároló-szervezés
  - pufferekkel való gazdálkodás

Átalakítás legfontosabb elvei:
- szelekciókat minél előbb végrehajtani
- projekciókat — " —
- műveletek csoportosítása, kevés menetszámú műveletekre törekvés

Költségek számítása:

Alapvető költségelem:
- a háttértároló blokkjainak
- olvasása a memóriába
- írása a memóriából

I/O költség

Művelet költségbecslésének paraméterei:
- operandusok mérete, sorok és blokkok számában: R reláció: T_R sor, B_R blokk.
- eredmény mérete (sorok, blokkok száma) nem optimalizálható
- művelet I/O költsége
- memória mérete blokkokban: M

(1)

Bui

Összefoglaló az alapműveletekről:

R mérete T_R sor, B_R blokk,

S mérete T_S sor, B_R blokk

M : szabadon használható memória mérete blokkokban

I_A : Az A attribútum előforduló értékeinek száma (képméret)

Művelet	Feltétel	Eredmény mérete	Eljárás költsége	Menetszám
$R \times S$	$B_R < M$	$T_R B_S + T_S B_R$	$B_R + B_S$	1
$R \times S$	$M < \min(B_R, B_S)$, $M < B_R \leq B_S$	$T_R B_S + T_S B_R$	$B_R + \frac{B_R B_S}{M-1}$	$\frac{B_R}{M-1}$
$R \bowtie_A S$	$M > \min(B_R, B_S)$	$\frac{T_R B_S + T_S B_R}{I_A}$	$B_R + B_S$	1
$R \bowtie_A S$,	rendezéses fájlok A-ra; „reális" méretű	$\frac{T_R B_S + T_S B_R}{I_A}$	összefuttatás: $B_R + B_S$	1
$R \bowtie_A S$,	+ rendezések elkészítése		+ rendezési költség: $2 B_R \log_M B_R + 2 B_S \log_M B_S$	

$\sigma_{F_1 \wedge \ldots \wedge F_k}(R)$: ki lehet használni F_i feltételt, ha pl. F_i : $A \theta$ "a"

$R \cup S$, $R - S$, $R \cap S$, $\Pi_{lista}(R)$: ha nem multihalmaz, rendezésben hasonló bonyolultságú, (n log n).

Multihalmaz: sorokhoz <u>számlálót</u> rendelünk, $R - S$ és $R \cap S$ esetén fontos! 1, 2 menetes kiértékelési feltétele!

②

Bunk Attila

Vizsgakérdések

1. Adatmodellezés, a relációs adatmodell
   I/1–4. oldal

2. Műveletek relációkkal, az algebrai lekérdezések
   I/5–7.

3. Fogalmi – logikai adatmodellezés:
   Az Egyed – Kapcsolat modell (ER modell)

4. Az Egyed – Kapcsolat modell leképezése a relációs modellbe; UML

5. Relációs adatbázisok tervezési elmélete:
   Funkcionális függőségek, többértékű függőségek, összekapcsolási függőségek, normál formák, normalizálás

6. Megszorítások az SQL-ben.
   Kulcsok, idegen kulcsok, CHECK, ASSERTION, TRIGGER

7. Tranzakciókezelés: 1.
   ACID elvárás, tranzakciós lépései, ütemezések, sorolhatóság, zárak

8. Tranzakciókezelés 2.
   Sorolhatóság ellenőrzése, kétfázisú zár-protokoll, hibák elleni védelem, naplók

9. Az objektum-relációs adatmodell 1.

10. Az objektum-relációs adatmodell 2.
    (Referenciák SQL99-ben, műveletek SQL 99-ben)

11. Fájkezelés, elsődleges, másodlagos indexek, B-fák

12. További indexek, lekérdezések kiértékelése